Simple Viewer Agent

CG

Contents

1 Introduction

This short text describes a system called the Simple Viewer Agent, or SVA. The system consists of a Java program which executes a small number of commands which collectively can be used to do some simple graphics processing, and another program, probably written in some other language, which sends graphics oriented messages to the Java program. In this document, the other program is taken to be a Lisp API, which presumably is being used in the service of some Lisp application.

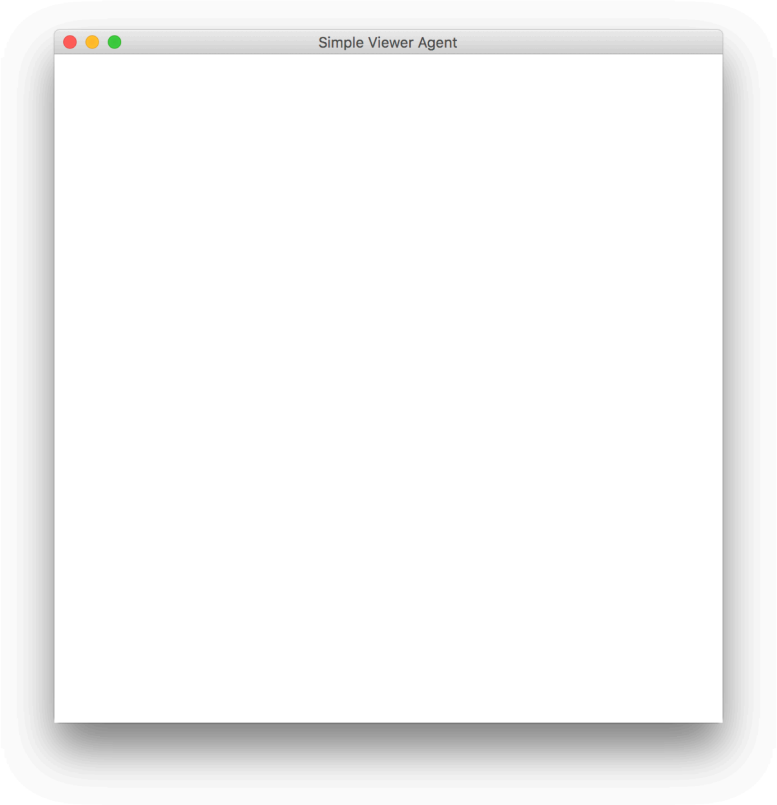
The Java program, `JavaViewerAgent.java`, a simple graphics rendering program, is detailed in Section 2 of this document. The Lisp API, `simple_viewer_agent.l`, is detailed in Section 3 of this document. Section 4 provides information on how to secure and run SVA. Sections 5 and 6 present the computer programs that make up SVA. The remaining sections present some suggestive examples of how the Simple Viewer Agent might be used.

As may be seen in Section 6, the Lisp API is a very short program. Analogous very short program may readily be written in other languages, e.g., Prolog, or Haskell, for those who might want to make use of SVA from another programming environment.

2 The Java Graphics Rendering Program

The Java graphics rendering program presents a frame, and renders graphics in the frame, relative to a seen or imagined grid of squares, in accordance with messages that it reads from a data file. The data file happens to be called `view.text`, and must be placed in the same directory as the Java program, but none of that needs to be known by the user of the system.

When the SVA Java program is run, a simple Frame filled with a blank canvas will appear on the screen. Hidden from view is a grid. No matter its size, for all practical purposes. The program that uses this rendering engine should always open with an initialization command which establishes the rendering environment.



The Java program processes 4 types of message:

- `INIT` messages specify the number of squares on each side of the grid of squares, whether or not the grid will be visible, and the amount of space, in pixels, that will separate rendered shapes from grid cell borders.
- `MODE` messages specify whether the shape to be rendered will be a circle or a square.
- `PAINT` messages specify a list of location/color pairs (conceptually) which indicate where shapes are to be painted, as well as the color of each shape that is to be painted. In this system, "to paint" means to render a filled in shape.
- `DRAW` messages specify a list of location/color pairs (conceptually) which indicate where shapes are to be drawn, as well as the color of each shape that is to be drawn. In this system, "to draw" means to render just the outline of a shape.

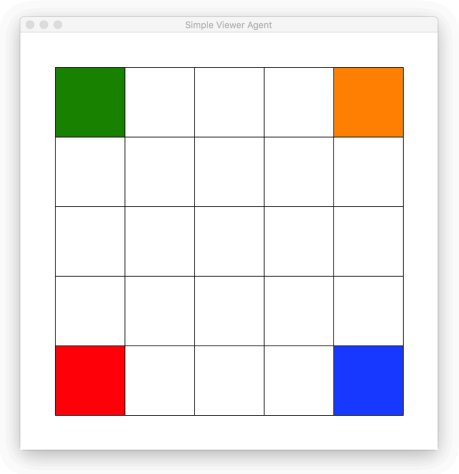
Details can be inferred from the examples which will soon follow.

Example 1

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 SHOW 0
SVA: message = MODE SQUARE
SVA: message = PAINT 1 1 R 1 5 I 5 1 G 5 5 0
```

Then the following graphic will be rendered:

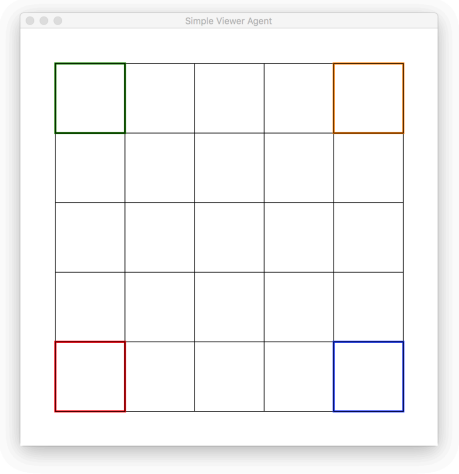


Example 2

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 SHOW 0
SVA: message = MODE SQUARE
SVA: message = DRAW 1 1 R 1 5 I 5 1 G 5 5 0
```

Then the following graphic will be rendered:

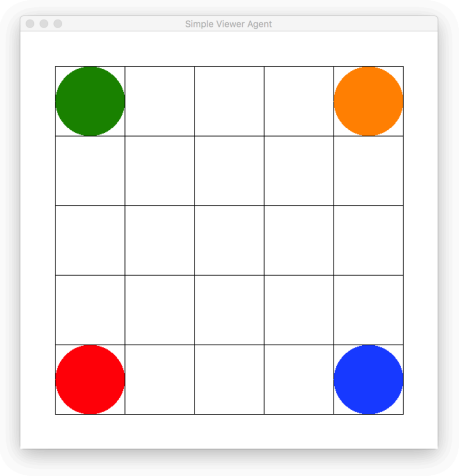


Example 3

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 SHOW 0
SVA: message = MODE CIRCLE
SVA: message = PAINT 1 1 R 1 5 I 5 1 G 5 5 0
```

Then the following graphic will be rendered:

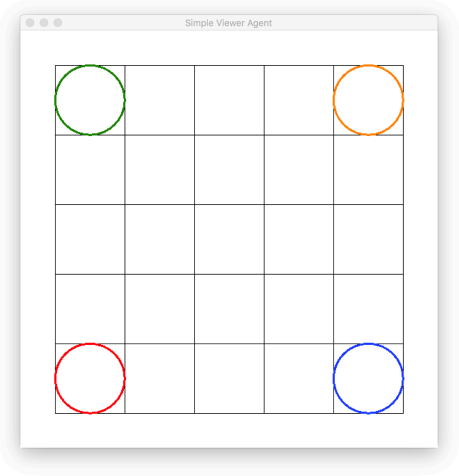


Example 4

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 SHOW 0
SVA: message = MODE CIRCLE
SVA: message = DRAW 1 1 R 1 5 I 5 1 G 5 5 0
```

Then the following graphic will be rendered:

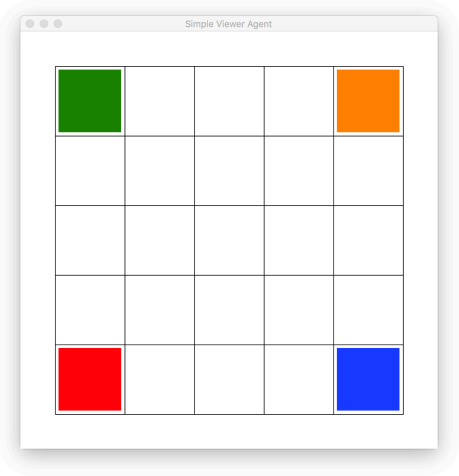


Example 5

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 SHOW 10
SVA: message = MODE SQUARE
SVA: message = PAINT 1 1 R 1 5 I 5 1 G 5 5 O
```

Then the following graphic will be rendered:

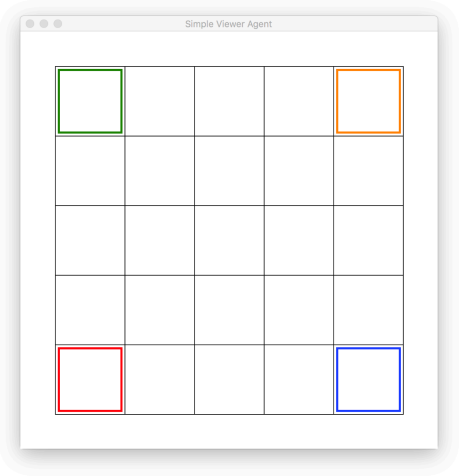


Example 6

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 SHOW 10
SVA: message = MODE SQUARE
SVA: message = DRAW 1 1 R 1 5 I 5 1 G 5 5 O
```

Then the following graphic will be rendered:

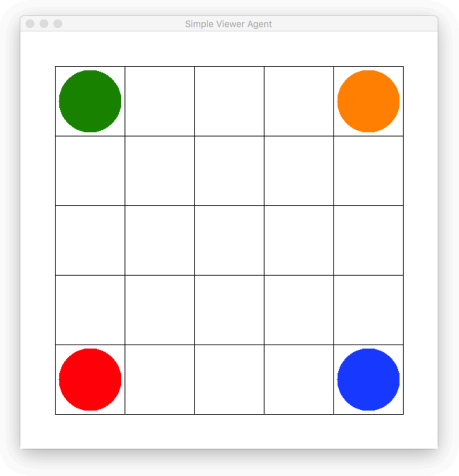


Example 7

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 SHOW 10
SVA: message = MODE CIRCLE
SVA: message = PAINT 1 1 R 1 5 I 5 1 G 5 5 O
```

Then the following graphic will be rendered:

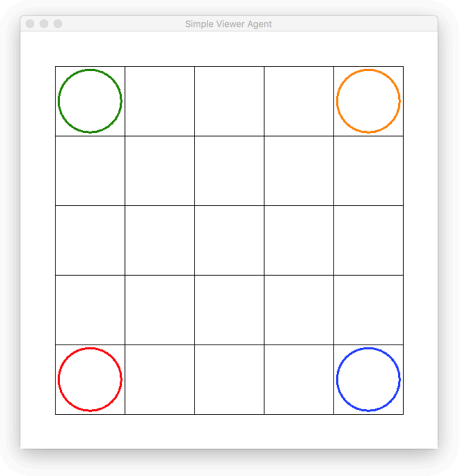


Example 8

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 SHOW 10
SVA: message = MODE CIRCLE
SVA: message = DRAW 1 1 R 1 5 I 5 1 G 5 5 O
```

Then the following graphic will be rendered:



Example 9

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 HIDE 0
SVA: message = MODE SQUARE
SVA: message = PAINT 1 1 A 1 2 B 1 3 C 1 4 D 1 5 E 2 1 F 2 2 G 2 3 H 2 4 I 2 5 J 3 1 K
3 2 L 3 3 M 3 4 N 3 5 O 4 1 P 4 2 Q 4 3 R 4 4 S 4 5 T 5 1 U 5 2 V 5 3 X 5 4 Y 5 5 Z
```

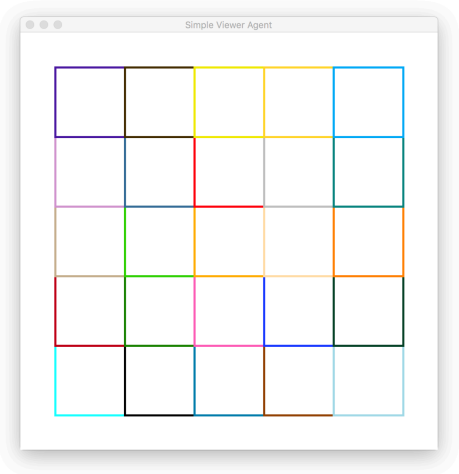
Then the following graphic will be rendered:

Example 10

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 HIDE 0
SVA: message = MODE SQUARE
SVA: message = DRAW 1 1 A 1 2 B 1 3 C 1 4 D 1 5 E 2 1 F 2 2 G 2 3 H 2 4 I 2 5 J 3 1 K
3 2 L 3 3 M 3 4 N 3 5 O 4 1 P 4 2 Q 4 3 R 4 4 S 4 5 T 5 1 U 5 2 V 5 3 X 5 4 Y 5 5 Z
```

Then the following graphic will be rendered:

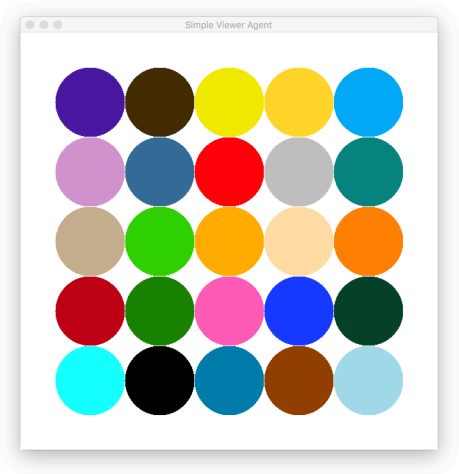


Example 11

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 HIDE 0
SVA: message = MODE CIRCLE
SVA: message = PAINT 1 1 A 1 2 B 1 3 C 1 4 D 1 5 E 2 1 F 2 2 G 2 3 H 2 4 I 2 5 J 3 1 K
3 2 L 3 3 M 3 4 N 3 5 O 4 1 P 4 2 Q 4 3 R 4 4 S 4 5 T 5 1 U 5 2 V 5 3 X 5 4 Y 5 5 Z
```

Then the following graphic will be rendered:

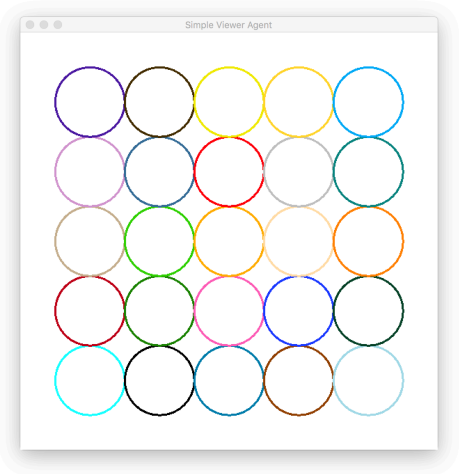


Example 12

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 HIDE 0
SVA: message = MODE CIRCLE
SVA: message = DRAW 1 1 A 1 2 B 1 3 C 1 4 D 1 5 E 2 1 F 2 2 G 2 3 H 2 4 I 2 5 J 3 1 K
3 2 L 3 3 M 3 4 N 3 5 O 4 1 P 4 2 Q 4 3 R 4 4 S 4 5 T 5 1 U 5 2 V 5 3 X 5 4 Y 5 5 Z
```

Then the following graphic will be rendered:

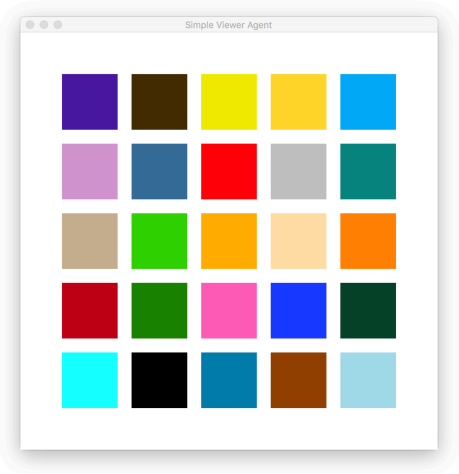


Example 13

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 HIDE 20
SVA: message = MODE SQUARE
SVA: message = PAINT 1 1 A 1 2 B 1 3 C 1 4 D 1 5 E 2 1 F 2 2 G 2 3 H 2 4 I 2 5 J 3 1 K
3 2 L 3 3 M 3 4 N 3 5 O 4 1 P 4 2 Q 4 3 R 4 4 S 4 5 T 5 1 U 5 2 V 5 3 X 5 4 Y 5 5 Z
```

Then the following graphic will be rendered:

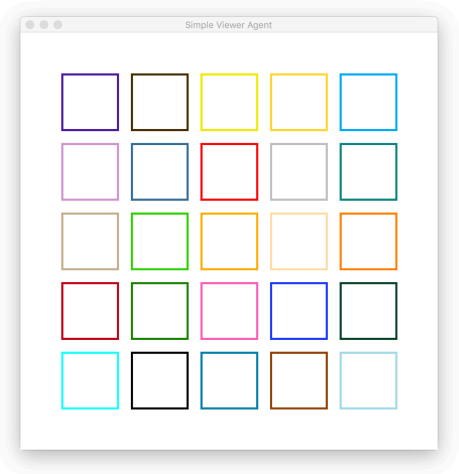


Example 14

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 HIDE 20
SVA: message = MODE SQUARE
SVA: message = DRAW 1 1 A 1 2 B 1 3 C 1 4 D 1 5 E 2 1 F 2 2 G 2 3 H 2 4 I 2 5 J 3 1 K
3 2 L 3 3 M 3 4 N 3 5 O 4 1 P 4 2 Q 4 3 R 4 4 S 4 5 T 5 1 U 5 2 V 5 3 X 5 4 Y 5 5 Z
```

Then the following graphic will be rendered:



Example 15

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 HIDE 20
SVA: message = MODE CIRCLE
SVA: message = PAINT 1 1 A 1 2 B 1 3 C 1 4 D 1 5 E 2 1 F 2 2 G 2 3 H 2 4 I 2 5 J 3 1 K
3 2 L 3 3 M 3 4 N 3 5 O 4 1 P 4 2 Q 4 3 R 4 4 S 4 5 T 5 1 U 5 2 V 5 3 X 5 4 Y 5 5 Z
```

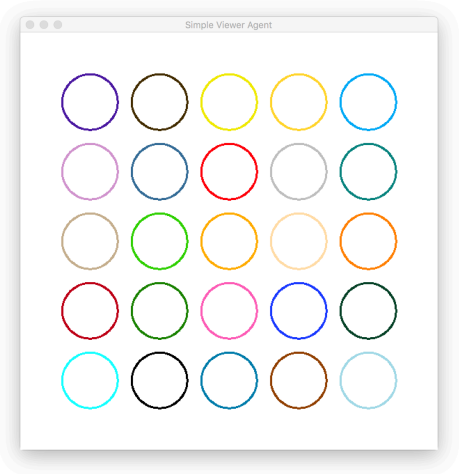
Then the following graphic will be rendered:

Example 16

Suppose the following messages are sent from Lisp:

```
SVA: message = INIT 5 HIDE 20
SVA: message = MODE CIRCLE
SVA: message = DRAW 1 1 A 1 2 B 1 3 C 1 4 D 1 5 E 2 1 F 2 2 G 2 3 H 2 4 I 2 5 J 3 1 K
3 2 L 3 3 M 3 4 N 3 5 O 4 1 P 4 2 Q 4 3 R 4 4 S 4 5 T 5 1 U 5 2 V 5 3 X 5 4 Y 5 5 Z
```

Then the following graphic will be rendered:



3 The Lisp API for SVA

The Java application SimpleViewerAgent.jar must be running in the directory from which your Lisp program is running for the SVA system to do its thing.

The Lisp API features just one essential bit of functionality, a "messenger", by which I mean a method which acts as a messaging agent. This method, called `send`, takes one string parameter, which is presumed to be bound to a command that can be interpreted by the Java graphics renderer, and writes the command/message to the file that is shared with the Java program.

In order to assure success with respect to the graphics processing, the Lisp API also includes four compilers, methods that translate list representations of commands to string representations of the commands, one for each command type. By inspecting the following slightly edited sample Lisp session, you will be able to infer the names of these compilers and the actions of translation that they perform.

```
bash-3.2$ clisp
...

[1]> ( load "sva_basic_demos.l" )
;; Loading file sva_basic_demos.l ...
;;  Loading file simple_viewer_agent.l ...
;;   Loading file ../rlp.l ...
;;   Loaded file ../rlp.l
;;  Loaded file simple_viewer_agent.l
;; Loaded file sva_basic_demos.l
T

[2]> ( trace send init mode paint draw )
;; Tracing function SEND.
;; Tracing function INIT.
;; Tracing function MODE.
;; Tracing function PAINT.
;; Tracing function DRAW.
(SEND INIT MODE PAINT DRAW)

[3]> ( demo-1 )
1. Trace: (INIT '(5 SHOW 20))
1. Trace: INIT ==> "INIT 5 SHOW 20"

1. Trace: (SEND '"INIT 5 SHOW 20")
1. Trace: SEND ==> NIL

1. Trace: (MODE 'CIRCLE)
1. Trace: MODE ==> "MODE CIRCLE"

1. Trace: (SEND '"MODE CIRCLE")
1. Trace: SEND ==> NIL

1. Trace: (PAINT '((1 1 R) (1 5 I) (5 1 G) (5 5 O)))
1. Trace: PAINT ==> "PAINT 1 1 R 1 5 I 5 1 G 5 5 O"
```

```
1. Trace: (SEND '"PAINT 1 1 R 1 5 I 5 1 G 5 5 O")
1. Trace: SEND ==> NIL
NIL

[4]> ( demo-4 )
1. Trace: (INIT '(7 SHOW 10))
1. Trace: INIT ==> "INIT 7 SHOW 10"

1. Trace: (SEND '"INIT 7 SHOW 10")
1. Trace: SEND ==> NIL

1. Trace: (MODE 'SQUARE)
1. Trace: MODE ==> "MODE SQUARE"

1. Trace: (SEND '"MODE SQUARE")
1. Trace: SEND ==> NIL

1. Trace: (DRAW '((1 1 R) (1 7 I) (7 1 G) (7 7 O)))
1. Trace: DRAW ==> "DRAW 1 1 R 1 7 I 7 1 G 7 7 O"

1. Trace: (SEND '"DRAW 1 1 R 1 7 I 7 1 G 7 7 O")
1. Trace: SEND ==> NIL
NIL

[5]>
```

Why the Compilers?

It is always best to catch errors as early as possible. This principle manifests in the idea that catching errors at compile time is far preferable to catching them at run time. (In time, Rust will replace C++ for just this reason! (Of course, entrenched senior software engineers may have to die off or be replaced for this to happen.))

The idea of the compilers in SVA is that they will help to assure that only executable commands are passed to the Java rendering engine. The degree to which they will help will depend on how much time I decide to devote to infusing them with integrity checking code. For now, they should be used in the spirit of compiler time checking, with the understanding that they are being offered in pre-release status.

Specification the Compilers

The following brief specifications are intended to be read with the sample session previously presented firmly in mind.

- `INIT` takes a list of three values as its sole parameter. The first in a positive integer representing the number of squares on each side of the grid. The second is the word `SHOW` or the word `HIDE`, which indicates whether or not the grid should be shown or hidden. The third indicates how much space should appear between a shape and the boundaries of its grid square.
- `MODE` takes a symbolic atom as its sole parameter. The word `CIRCLE` indicates that circles will be rendered. The word `SQUARE` indicates that squares will be rendered.
- `PAINT` takes a list of triples as its sole parameter. The first two components of a triple represent a location on the grid. The last component represents a color by means of a symbolic atom.
- `DRAW` takes a list of triples as its sole parameter. The first two components of a triple represent a location on the grid. The last component represents a color by means of a symbolic atom.

The Color Map

- `A` → aqua (0,255,255)
- `B` → black (0,0,0)
- `C` → cerulean (0,123,167)
- `D` → dandelion (254,216,93)
- `E` → eucalyptus (173,216,230)
- `F` → firebrick (178,34,34)
- `G` → green (0,128,0)
- `H` → hotpink (255,105,180)
- `I` → indigo (75,0,130)
- `J` → jade (0,66,42)
- `K` → khaki (195,176,145)
- `L` → lime (50,205,50)
- `M` → mango (254,180,15)
- `N` → navajowhite (255,222,173)
- `O` → orange (255,140,0)
- `P` → plum (204,153,204)
- `Q` → queenblue (67,107,149)
- `R` → red (255,0,0)
- `S` → silver (192,192,192)
- `T` → teal (0,128,128)
- `U` → ultraviolet (75,1,156)
- `V` → vandykebrown (65,48,0)
- `W` → white (255,255,255)
- `X` → xanthic (238,237,9)
- `Y` → yellow (255,219,88)
- `Z` → zomp (57,167,142)

4 Getting and Using SVA

1. Establish a directory in which to work. The SVA Java program and the SVA Lisp program must run from within this directory.
2. Go to the following site: `http://www.cs.oswego.edu/ blue/software/`
3. Download the following files:
   - `SimpleViewerAgent.jar`
   - `simple_viewer_agent.l`
   - `sva_basic_demos.l`
4. Run the `SimpleViewerAgent.jar` file from a command line, by typing something like:
   `java -jar SimpleViewerAgent.jar`
5. Run Lisp from somewhere else, like an emacs shell, or another terminal window, and load the file of Lisp demos, which in turn will load the Lisp API.
6. Try a demo, perhaps enter `( demo-9 )` at the Lisp prompt.

Trace of the Using

The Shell for the Java Agent

```
bash-3.2$ ls
SimpleViewerAgent.jar simple_viewer_agent.l sva_basic_demos.l
bash-3.2$ java -jar SimpleViewerAgent.jar
SVA: message = INIT 5 HIDE 0
SVA: message = MODE SQUARE
SVA: message = PAINT 1 1 A 1 2 B 1 3 C 1 4 D 1 5 E 2 1 F 2 2 G 2 3 H 2 4 I 2 5 J 3 1 K 3 2 L 3 3 M
3 4 N 3 5 O 4 1 P 4 2 Q 4 3 R 4 4 S 4 5 T 5 1 U 5 2 V 5 3 X 5 4 Y 5 5 Z
```

The Shell for the Lisp Process

```
bash-3.2$ clisp
...

[1]> ( load "sva_basic_demos.l" )
;; Loading file sva_basic_demos.l ...
;;  Loading file simple_viewer_agent.l ...
;;  Loaded file simple_viewer_agent.l
;; Loaded file sva_basic_demos.l
T
[2]> ( demo-9 )
```

```
NIL
[3]>
```

The demo-9 Code

```
( defun demo-9 ()
  ( send ( init ( list 5 'hide 0 ) ) )
  ( send ( mode 'square ) )
  ( send ( paint '(
    ( 1 1 A ) ( 1 2 B ) ( 1 3 C ) ( 1 4 D ) ( 1 5 E )
    ( 2 1 F ) ( 2 2 G ) ( 2 3 H ) ( 2 4 I ) ( 2 5 J )
    ( 3 1 K ) ( 3 2 L ) ( 3 3 M ) ( 3 4 N ) ( 3 5 O )
    ( 4 1 P ) ( 4 2 Q ) ( 4 3 R ) ( 4 4 S ) ( 4 5 T )
    ( 5 1 U ) ( 5 2 V ) ( 5 3 X ) ( 5 4 Y ) ( 5 5 Z )
  ) ) )
  nil
)
```

The Canvas

5 Java Graphics Rendering Source

```
/*
 * Program designed to repeatedly:
 * (1) spot the existence of a file called view.text containing one of four
 *     commands, either PAINT or DRAW or MODE or INIT
 * (2) execute the command found in the file
 * (3) remove the view.text file
 */

package simplevieweragent;

import java.awt.Color;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.Scanner;
import javax.swing.SwingUtilities;
import painter.SPainter;
import shapes.SCircle;
import shapes.SSquare;

/**
 *
 * @author blue
 */
public class SimpleViewerAgent {

    // THE CONSTRUCTOR MERELY READIES THE ENVIRONMENT

    public SimpleViewerAgent() {
        establishPainter();
        initColorMap();
    }

    // THE MAIN METHOD PRESENTS THE PAINTER AND THEN REPEATEDLY INTERPRETS
    // A COMMAND WHEN FOUND IN THE DESIGNATED FILE

    public static void main(String[] args) throws FileNotFoundException {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new SimpleViewerAgent();
            }
        });
        if ( fileExistsP() ) { deleteTheFile(); }
        for (;;) {
           if ( fileExistsP() ) {
               processTheFile();
               deleteTheFile();
```

```
       }
    }
}

static private String userDir = System.getProperty("user.dir");
static private String fileName = userDir + "/" + "view.text";

static private boolean fileExistsP() {
    try {
        File f = new File(fileName);
        return f.exists();
    } catch ( Exception e ) {
        return false;
    }
}

static private void deleteTheFile() {
    try {
        File f = new File(fileName);
        f.delete();
    } catch ( Exception e ) {
        System.out.println("### trouble deleting the file.");
    }

}

// THIS IS THE INTERPRETER

static private void processTheFile() throws FileNotFoundException  {
    Scanner scanner = new Scanner(new FileReader(fileName));
    String command = scanner.next();
    String line = "";
    while ( scanner.hasNext() ) {
        String token = scanner.next();
        line = line + token + " ";
    }
    line = line.trim();
    String message = command + " " + line;
    System.out.println("SVA: message = " + message);
    if ( command.equalsIgnoreCase("PAINT") ) {
        paint(line);
    } else if ( command.equalsIgnoreCase("DRAW") ) {
        draw(line);
    } else if ( command.equalsIgnoreCase("MODE") ) {
        mode(line);
    } else if ( command.equalsIgnoreCase("INIT") ) {
        init(line);
    } else {
        System.out.println("SVA: I cannot do this: " + command);
    }
}

// PAINT MEANS RENDER THE INTERIOR OF THE SHAPE
```

```
private static void paint(String line) {
    Scanner scanner = new Scanner(line);
    while ( scanner.hasNext() ) {
        int row = scanner.nextInt();
        int col = scanner.nextInt();
        String colorCode = scanner.next();
        paint(row,col,convert(colorCode));
    }
}

static private void paint(int row, int col, Color color ) {
    painter.mfd(gridSquare.side()*(row-1));
    painter.mrt(gridSquare.side()*(col-1));
    painter.setColor(color);
    if ( drawMode.equalsIgnoreCase("SQUARE") ) {
        painter.paint(square);
        if ( gridMode.equalsIgnoreCase("SHOW") ) {
            painter.setColor(Color.BLACK);
            painter.draw(gridSquare);
        }
    } else if ( drawMode.equalsIgnoreCase("CIRCLE") ) {
        painter.paint(circle);
    }
    painter.mbk(gridSquare.side()*(row-1));
    painter.mlt(gridSquare.side()*(col-1));
}

// DRAW MEANS RENDER THE BORDER OF THE SHAPE

private static void draw(String line) {
    Scanner scanner = new Scanner(line);
    while ( scanner.hasNext() ) {
        int row = scanner.nextInt();
        int col = scanner.nextInt();
        String colorCode = scanner.next();
        draw(row,col,convert(colorCode));
    }
}

static private void draw(int row, int col, Color color ) {
    painter.mfd(gridSquare.side()*(row-1));
    painter.mrt(gridSquare.side()*(col-1));
    painter.setColor(color);
    if ( drawMode.equalsIgnoreCase("SQUARE") ) {
        painter.setBrushWidth(3);
        painter.draw(square);
        painter.setBrushWidth(1);
        if ( gridMode.equalsIgnoreCase("SHOW") ) {
            painter.setColor(Color.BLACK);
            painter.draw(gridSquare);
        }
    } else if ( drawMode.equalsIgnoreCase("CIRCLE") ) {
        painter.setBrushWidth(3);
        painter.draw(circle);
```

```
        painter.setBrushWidth(1);
    }
    painter.mbk(gridSquare.side()*(row-1));
    painter.mlt(gridSquare.side()*(col-1));
}

// ENTER circle MODE OR square MODE
private static void mode(String line) {
    Scanner scanner = new Scanner(line);
    drawMode = scanner.next();
    // SOMEDAY, CHECK FOR circle OR square (EVENTUALLY text OR line, TO)
}

// TAILORS THE ENVIRONMENT TO FIT THE CURRENT NEED, AND CREATES
// CORRESPONDING OBJECTS TO WORK WITH

static private void init(String line) {
    Scanner scanner = new Scanner(line);
    n = scanner.nextInt();
    cellSide = gridSide / n;
    gridMode = scanner.next();
    spacer = scanner.nextInt();
    // !!! VERIFY SHOW OR HIDE
    ready();
}

static private void ready() {
    int radius = ( cellSide / 2 ) - ( spacer / 2 );
    circle = new SCircle(radius);
    int side = cellSide - spacer;
    square = new SSquare(side);
    int gridSide = cellSide;
    gridSquare = new SSquare(gridSide);
    eraser = new SSquare(1000);
    painter.moveToCenter();
    painter.setColor(Color.WHITE);
    painter.paint(eraser);
    for ( int i = 1; i <= n-1; i++ ) {
        painter.mbk(gridSquare.side()/2);
        painter.mlt(gridSquare.side()/2);
    }
    if ( gridMode.equalsIgnoreCase("SHOW") ) { grid(); }
}

static private void grid() {
    painter.setColor(Color.BLACK);
    for ( int i = 1; i <= n; i++ ) {
        line();
        painter.mrt(gridSquare.side());
    }
    for ( int i = 1; i <= n; i++ ) {
        painter.mlt(gridSquare.side());
    }
}
```

```
static private void line () {
    for ( int i = 1; i <= n; i++ ) {
        painter.draw(gridSquare);
        painter.mfd(gridSquare.side());
    }
    for ( int i = 1; i <= n; i++ ) {
        painter.mbk(gridSquare.side());
    }
}

// ESTABLISH THE PAINTER AND ENVIRONMENTAL PARAMETERS

private static final int gridSide = 500;
static private int spacer = 0;
static private int n = 10;
static private int cellSide = gridSide / n;
static String gridMode = "hide";
static String drawMode = "circle";
static SPainter painter;
static SCircle circle;
static SSquare square;
static SSquare gridSquare;
static SSquare eraser;

static private void establishPainter() {
    painter = new SPainter("Simple Viewer Agent",600,600);
    ready();
}

// COLOR MAPPING CODE

static private Color convert(String colorCode) {
    for ( int x = 0; x < colorCodes.size(); x++ ) {
        if ( colorCodes.get(x).equalsIgnoreCase(colorCode) ) {
            return colors.get(x);
        }
    }
    return Color.GRAY;
}

static private ArrayList<String> colorCodes = new ArrayList<>();
static private ArrayList<Color> colors = new ArrayList<>();

static private void initColorMap() {
    colorCodes.add("A"); // aqua
    colorCodes.add("B"); // black
    colorCodes.add("C"); // cerulean
    colorCodes.add("D"); // dirt
    colorCodes.add("E"); // eucalyptus
    colorCodes.add("F"); // firebrick
    colorCodes.add("G"); // green
    colorCodes.add("H"); // hotpink
    colorCodes.add("I"); ///indigo
```

```
        colorCodes.add("J"); // jade
        colorCodes.add("K"); // khaki
        colorCodes.add("L"); // lime
        colorCodes.add("M"); // mango
        colorCodes.add("N"); // navajowhite
        colorCodes.add("O"); // orange
        colorCodes.add("P"); // plum
        colorCodes.add("Q"); // queenblue
        colorCodes.add("R"); // red
        colorCodes.add("S"); // silver
        colorCodes.add("T"); // teal
        colorCodes.add("U"); // ultraviolet
        colorCodes.add("V"); // vandykebrown
        colorCodes.add("W"); // white
        colorCodes.add("X"); // xanthic
        colorCodes.add("Y"); // yellow
        colorCodes.add("Z"); // zomp
        colors.add(new Color(0,255,255));
        colors.add(new Color(0,0,0));
        colors.add(new Color(0,123,167));
        colors.add(new Color(139,69,19));
        colors.add(new Color(173,216,230));
        colors.add(new Color(178,34,34));
        colors.add(new Color(0,128,0));
        colors.add(new Color(255,105,180));
        colors.add(new Color(51,51,255));
        colors.add(new Color(0,66,42));
        colors.add(new Color(195,176,145));
        colors.add(new Color(50,205,50));
        colors.add(new Color(254,180,15));
        colors.add(new Color(255,222,173));
        colors.add(new Color(255,140,0));
        colors.add(new Color(204,153,204));
        colors.add(new Color(67,107,149));
        colors.add(new Color(255,0,0));
        colors.add(new Color(192,192,192));
        colors.add(new Color(0,128,128));
        colors.add(new Color(75,1,156));
        colors.add(new Color(65,48,0));
        colors.add(new Color(255,255,255));
        colors.add(new Color(238,237,9));
        colors.add(new Color(255,219,88));
        colors.add(new Color(57,167,242));
    }

}
```

6 Lisp API Source

```
;;;; File: simple_viewer_agent.l
;;;; Line: Lisp API for doing just a few very simple visual things
;;;; More: See the Simple Viewer Agent guide and report

;; THE FEATURED FUNCTIONALITY -- a messenger and four compilers

( defun send (command)
  ( write-to-view-file command )
  ( sleep 1)
)

( defun draw ( triples )
  ( concatenate 'string "DRAW " ( list-to-string-proper ( flatten triples ) ) )
)

( defun paint ( triples )
  ( concatenate 'string "PAINT " ( list-to-string-proper ( flatten triples ) ) )
)

( defun mode ( mode )
  ( concatenate 'string "MODE " ( list-to-string-proper ( list mode ) ) )
)

( defun init ( triple )
  ( concatenate 'string "INIT " ( list-to-string-proper triple ) )
)

;; INFRASTRUCTURE FOR THE FEATURED FUNCTIONALITY

; simply create a string from the list, with the parentheses removed
( defun list-to-string-proper ( the-list )
  ( string-right-trim ")" ( string-left-trim "(" ( write-to-string the-list) ) )
)

; write a command to the designated output file
( defun write-to-view-file ( command )
  ( with-open-file
    ( *standard-output* "view.text" :direction :output :if-exists :supersede )
    ( format t "~A~%" command )
  )
)

;; UTILITIES

;; return a list of pairs representing the locations on an nxn grid
( defun locations (n)
  ( mapcan
    ( lambda (list1 list2) ( mapcar #'list list1 list2 ) )
```

```
    ( mapcar #'duplicate ( iota n ) ( duplicate n n ) )
    ( duplicate ( iota n ) n )
  )
)

;; return a list of triples for rendering an entire nxn grid in some color
( defun color-grid (n c)
  ( mapcar #'snoc ( duplicate c ( * n n ) ) ( locations n ) )
)

;; selected list processors

( defun snoc ( o l )
  ( cond
    ( ( null l )
      ( list o )
    )
    ( t
      ( cons ( car l ) ( snoc o ( cdr l ) ) )
    )
  )
)

( defun duplicate ( o n )
  ( cond
    ( ( = n 0 )
      ()
    )
    ( t
      ( cons o ( duplicate o ( - n 1 ) ) )
    )
  )
)

( defun iota ( n )
  ( cond
    ( ( = n 0 )
      ()
    )
    ( t
      ( snoc n ( iota ( - n 1 ) ) )
    )
  )
)

( defun take-from ( x l )
  ( filter-out ( lambda ( y ) ( equal x y ) ) l )
)

( defun pick ( l )
  ( nth ( random ( length l ) ) l )
)

( defun random-permutation ( l )
```

```
  ( cond
    ( ( null l )
      ()
    )
    ( t
      ( setf element ( pick l ) )
      ( setf remainder ( take-from element l ) )
      ( cons element ( random-permutation remainder ) )
    )
  )
)

( defun flatten (l)
  ( cond
    ( ( null l )
      ()
    )
    ( ( atom ( car l ) )
      ( cons ( car l ) ( flatten ( cdr l ) ) )
    )
    ( t
      ( append ( flatten ( car l ) ) ( flatten ( cdr l ) ) )
    )
  )
)
```

7 SVA Basic Lisp Demos

```
;;;; File: sva_basic_demos.l
;;;; Line: Basic demos for the Lisp API which serves to do a few very simple visual things

;; IMPORTS

( load "simple_viewer_agent.l" )

;; THE DEMOS

( defun demo-1 ()
  ( send ( init ( list 5 'show 0 ) ) )
  ( send ( mode 'square ) )
  ( send ( paint '( ( 1 1 R ) ( 1 5 I ) ( 5 1 G ) ( 5 5 O ) ) ) )
  nil
)

( defun demo-2 ()
  ( send ( init ( list 5 'show 0 ) ) )
  ( send ( mode 'square ) )
  ( send ( draw '( ( 1 1 R ) ( 1 5 I ) ( 5 1 G ) ( 5 5 O ) ) ) )
  nil
)

( defun demo-3 ()
  ( send ( init ( list 5 'show 0 ) ) )
  ( send ( mode 'circle ) )
  ( send ( paint '( ( 1 1 R ) ( 1 5 I ) ( 5 1 G ) ( 5 5 O ) ) ) )
  nil
)

( defun demo-4 ()
  ( send ( init ( list 5 'show 0 ) ) )
  ( send ( mode 'circle ) )
  ( send ( draw '( ( 1 1 R ) ( 1 5 I ) ( 5 1 G ) ( 5 5 O ) ) ) )
  nil
)

( defun demo-5 ()
  ( send ( init ( list 5 'show 10 ) ) )
  ( send ( mode 'square ) )
  ( send ( paint '( ( 1 1 R ) ( 1 5 I ) ( 5 1 G ) ( 5 5 O ) ) ) )
  nil
)

( defun demo-6 ()
  ( send ( init ( list 5 'show 10 ) ) )
  ( send ( mode 'square ) )
  ( send ( draw '( ( 1 1 R ) ( 1 5 I ) ( 5 1 G ) ( 5 5 O ) ) ) )
```

```
  nil
)

( defun demo-7 ()
  ( send ( init ( list 5 'show 10 ) ) )
  ( send ( mode 'circle ) )
  ( send ( paint '( ( 1 1 R ) ( 1 5 I ) ( 5 1 G ) ( 5 5 O ) ) ) )
  nil
)

( defun demo-8 ()
  ( send ( init ( list 5 'show 10 ) ) )
  ( send ( mode 'circle ) )
  ( send ( draw '( ( 1 1 R ) ( 1 5 I ) ( 5 1 G ) ( 5 5 O ) ) ) )
  nil
)

( defun demo-9 ()
  ( send ( init ( list 5 'hide 0 ) ) )
  ( send ( mode 'square ) )
  ( send ( paint '(
    ( 1 1 A ) ( 1 2 B ) ( 1 3 C ) ( 1 4 D ) ( 1 5 E )
    ( 2 1 F ) ( 2 2 G ) ( 2 3 H ) ( 2 4 I ) ( 2 5 J )
    ( 3 1 K ) ( 3 2 L ) ( 3 3 M ) ( 3 4 N ) ( 3 5 O )
    ( 4 1 P ) ( 4 2 Q ) ( 4 3 R ) ( 4 4 S ) ( 4 5 T )
    ( 5 1 U ) ( 5 2 V ) ( 5 3 X ) ( 5 4 Y ) ( 5 5 Z )
  ) ) )
  nil
)

( defun demo-10 ()
  ( send ( init ( list 5 'hide 0 ) ) )
  ( send ( mode 'square ) )
  ( send ( draw '(
    ( 1 1 A ) ( 1 2 B ) ( 1 3 C ) ( 1 4 D ) ( 1 5 E )
    ( 2 1 F ) ( 2 2 G ) ( 2 3 H ) ( 2 4 I ) ( 2 5 J )
    ( 3 1 K ) ( 3 2 L ) ( 3 3 M ) ( 3 4 N ) ( 3 5 O )
    ( 4 1 P ) ( 4 2 Q ) ( 4 3 R ) ( 4 4 S ) ( 4 5 T )
    ( 5 1 U ) ( 5 2 V ) ( 5 3 X ) ( 5 4 Y ) ( 5 5 Z )
  ) ) )
  nil
)

( defun demo-11 ()
  ( send ( init ( list 5 'hide 0 ) ) )
  ( send ( mode 'circle ) )
  ( send ( paint '(
    ( 1 1 A ) ( 1 2 B ) ( 1 3 C ) ( 1 4 D ) ( 1 5 E )
    ( 2 1 F ) ( 2 2 G ) ( 2 3 H ) ( 2 4 I ) ( 2 5 J )
    ( 3 1 K ) ( 3 2 L ) ( 3 3 M ) ( 3 4 N ) ( 3 5 O )
    ( 4 1 P ) ( 4 2 Q ) ( 4 3 R ) ( 4 4 S ) ( 4 5 T )
    ( 5 1 U ) ( 5 2 V ) ( 5 3 X ) ( 5 4 Y ) ( 5 5 Z )
  ) ) )
  nil
```

```
)

( defun demo-12 ()
  ( send ( init ( list 5 'hide 0 ) ) )
  ( send ( mode 'circle ) )
  ( send ( draw '(
    ( 1 1 A ) ( 1 2 B ) ( 1 3 C ) ( 1 4 D ) ( 1 5 E )
    ( 2 1 F ) ( 2 2 G ) ( 2 3 H ) ( 2 4 I ) ( 2 5 J )
    ( 3 1 K ) ( 3 2 L ) ( 3 3 M ) ( 3 4 N ) ( 3 5 O )
    ( 4 1 P ) ( 4 2 Q ) ( 4 3 R ) ( 4 4 S ) ( 4 5 T )
    ( 5 1 U ) ( 5 2 V ) ( 5 3 X ) ( 5 4 Y ) ( 5 5 Z )
  ) ) )
  nil
)

( defun demo-13 ()
  ( send ( init ( list 5 'hide 20 ) ) )
  ( send ( mode 'square ) )
  ( send ( paint '(
    ( 1 1 A ) ( 1 2 B ) ( 1 3 C ) ( 1 4 D ) ( 1 5 E )
    ( 2 1 F ) ( 2 2 G ) ( 2 3 H ) ( 2 4 I ) ( 2 5 J )
    ( 3 1 K ) ( 3 2 L ) ( 3 3 M ) ( 3 4 N ) ( 3 5 O )
    ( 4 1 P ) ( 4 2 Q ) ( 4 3 R ) ( 4 4 S ) ( 4 5 T )
    ( 5 1 U ) ( 5 2 V ) ( 5 3 X ) ( 5 4 Y ) ( 5 5 Z )
  ) ) )
  nil
)

( defun demo-14 ()
  ( send ( init ( list 5 'hide 20 ) ) )
  ( send ( mode 'square ) )
  ( send ( draw '(
    ( 1 1 A ) ( 1 2 B ) ( 1 3 C ) ( 1 4 D ) ( 1 5 E )
    ( 2 1 F ) ( 2 2 G ) ( 2 3 H ) ( 2 4 I ) ( 2 5 J )
    ( 3 1 K ) ( 3 2 L ) ( 3 3 M ) ( 3 4 N ) ( 3 5 O )
    ( 4 1 P ) ( 4 2 Q ) ( 4 3 R ) ( 4 4 S ) ( 4 5 T )
    ( 5 1 U ) ( 5 2 V ) ( 5 3 X ) ( 5 4 Y ) ( 5 5 Z )
  ) ) )
  nil
)

( defun demo-15 ()
  ( send ( init ( list 5 'hide 20 ) ) )
  ( send ( mode 'circle ) )
  ( send ( paint '(
    ( 1 1 A ) ( 1 2 B ) ( 1 3 C ) ( 1 4 D ) ( 1 5 E )
    ( 2 1 F ) ( 2 2 G ) ( 2 3 H ) ( 2 4 I ) ( 2 5 J )
    ( 3 1 K ) ( 3 2 L ) ( 3 3 M ) ( 3 4 N ) ( 3 5 O )
    ( 4 1 P ) ( 4 2 Q ) ( 4 3 R ) ( 4 4 S ) ( 4 5 T )
    ( 5 1 U ) ( 5 2 V ) ( 5 3 X ) ( 5 4 Y ) ( 5 5 Z )
  ) ) )
  nil
)
```

```
( defun demo-16 ()
  ( send ( init ( list 5 'hide 20 ) ) )
  ( send ( mode 'circle ) )
  ( send ( draw '(
    ( 1 1 A ) ( 1 2 B ) ( 1 3 C ) ( 1 4 D ) ( 1 5 E )
    ( 2 1 F ) ( 2 2 G ) ( 2 3 H ) ( 2 4 I ) ( 2 5 J )
    ( 3 1 K ) ( 3 2 L ) ( 3 3 M ) ( 3 4 N ) ( 3 5 O )
    ( 4 1 P ) ( 4 2 Q ) ( 4 3 R ) ( 4 4 S ) ( 4 5 T )
    ( 5 1 U ) ( 5 2 V ) ( 5 3 X ) ( 5 4 Y ) ( 5 5 Z )
  ) ) )
  nil
)

( defun demo-a ()  ; alphabet of colors
  ( send ( init ( list 1 'show 0 ) ) )
  ( send ( mode 'square ) )
  ( dolist ( letter '(a b c d e f g h i j k l m n o p q r s t u v w x y z) )
    ( send ( paint ( list ( list 1 1 letter ) ) ) )
    ( sleep 5 )
  )
  nil
)
```

8 Example: Rendering Dobo Graphically

The Scenario

The `sva_dobo_demo.l` file contains a demo for a dobo playing shell. It makes use of `sva.l`, the Lisp API for rendering graphics, which itself leans on the Java program `SimpleViewerAgent.java` which is contained in `SimpleViewerAgent.jar`.

Lisp Session

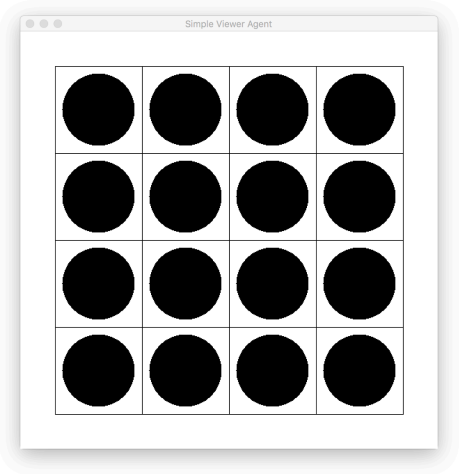
```
bash-3.2$ clisp
...

[1]> ( load "sva_dobo_demo.l" )
;; Loading file sva_dobo_demo.l ...
;;  Loading file simple_viewer_agent.l ...
;;  Loaded file simple_viewer_agent.l
;; Loaded file sva_dobo_demo.l
T
[2]> ( dobo-demo 4 )
Name of first player? apple
Name of second player? orange
Your move, APPLE ( ( 1 1 ) ( 2 1 ) ( 3 1 ) )
Your move, ORANGE ( ( 2 3 ) ( 3 3 ) )
Your move, APPLE ( ( 4 1 ) ( 4 2 ) ( 4 3 ) ( 4 4 ) )
Your move, ORANGE ( ( 1 2 ) ( 1 3 ) ( 1 4 ) )
Your move, APPLE ( ( 2 2 ) )
Your move, ORANGE ( ( 2 4 ) ( 3 4 ) )
Your move, APPLE ( ( 3 2 ) )
Congratulations, ORANGE!
NIL
[3]>
```

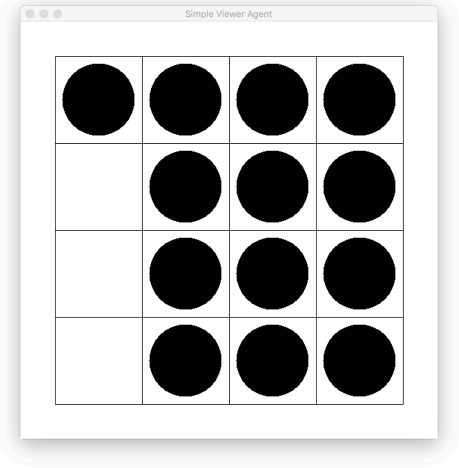
Textual Output from the Java Program

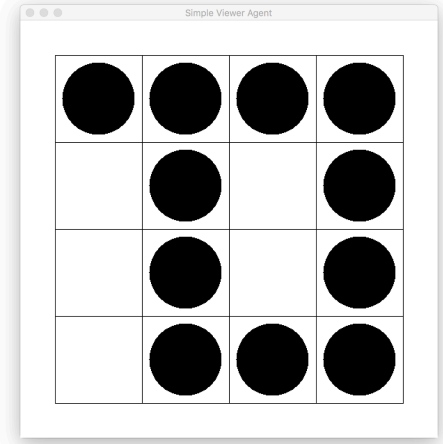
```
bash-3.2$ java -jar SimpleViewerAgent.jar
SVA: message = INIT 4 SHOW 20
SVA: message = MODE CIRCLE
SVA: message = PAINT 1 1 B 1 2 B 1 3 B 1 4 B 2 1 B 2 2 B 2 3 B 2 4 B 3 1 B
3 2 B 3 3 B 3 4 B 4 1 B 4 2 B 4 3 B 4 4 B
SVA: message = PAINT 1 1 W 2 1 W 3 1 W
SVA: message = PAINT 2 3 W 3 3 W
SVA: message = PAINT 4 1 W 4 2 W 4 3 W 4 4 W
SVA: message = PAINT 1 2 W 1 3 W 1 4 W
SVA: message = PAINT 2 2 W
```

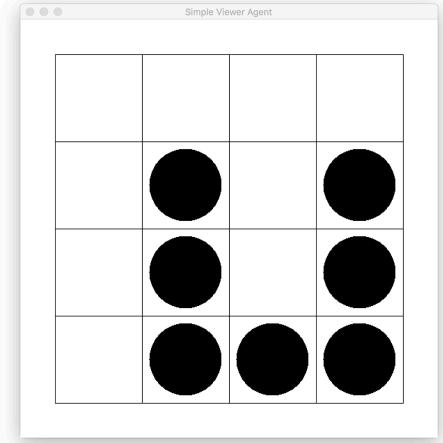
```
SVA: message = PAINT 2 4 W 3 4 W
SVA: message = PAINT 3 2 W
```

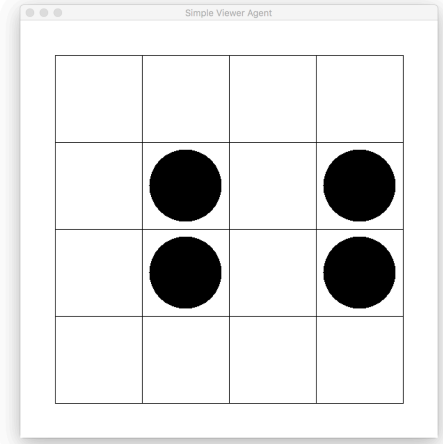
Snapshots of the Graphical Output

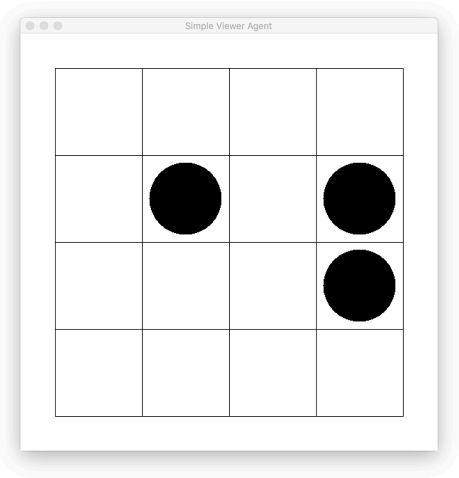


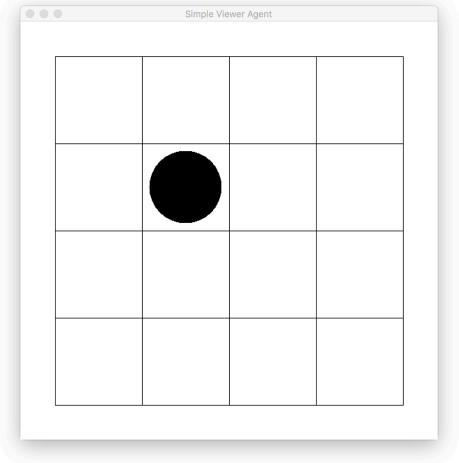


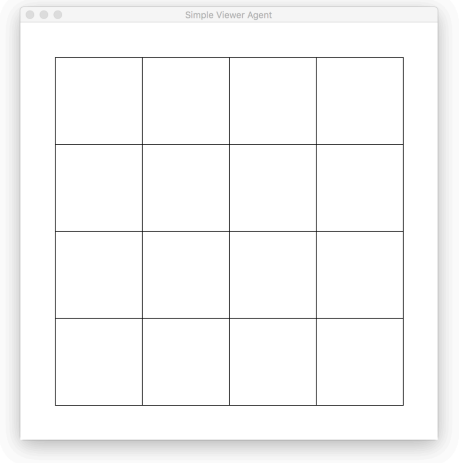

Simple Viewer Agent


Simple Viewer Agent


Simple Viewer Agent


Simple Viewer Agent


Simple Viewer Agent


Simple Viewer Agent

sva_dobo_demo.l

```
;;;; File: sva_dobo_demo.l
;;;; Line: Dobo demo of the Lisp API for doing just a few very simple visual things

;; IMPORTS

( load "simple_viewer_agent.l" )

;; THE DOBO DEMO OF sva.l

( defun dobo-demo (n)
  ( send ( init ( list n 'show ) ) )
  ( send ( mode 'circle ) )
  ( send ( draw ( color-grid n 'b ) ) )
  ( multiple-value-setq (first second) (get-player-names) )
  ( setf stones ( locations n ) )
  ( play-dobo-game first second stones )
  nil
)

( defun get-player-names (&aux first second)
  ( format t "Name of first player? " )
  ( setf first ( read ) )
  ( format t "Name of second player? " )
  ( setf second ( read ) )
  ( values first second )
)

( defun play-dobo-game (current next stones)
  ( format t "Your move, ~A " current )
  ( setf pairs ( read ) )
  ( send ( draw ( mapcar #'snoc ( duplicate 'w ( length pairs ) ) pairs ) ) )
  ( setf stones ( remove-pairs pairs stones ) )
  ( if ( null stones )
      ( let ()
        ( format t "Congratulations, ~A!~%" next )
        ( return-from play-dobo-game nil )
      )
  )
  ( play-dobo-game next current stones )
)

( defun remove-pairs (pairs stones)
  ( cond
    ( ( null pairs ) stones )
    ( t
      ( remove-pairs ( cdr pairs ) ( remove ( car pairs ) stones :test #'equal ) )
    )
  )
)
```

9 Example: Rendering Peg Solitaire Graphically

The Scenario

The `sva_pegs_demo.l` file contains a demo for peg solitaire. It makes use of `simple_viewer_agent.l`, the Lisp API for rendering graphics, which itself leans on the Java program `SimpleViewerAgent.java` which is contained in `SimpleViewerAgent.jar`.

Lisp Session

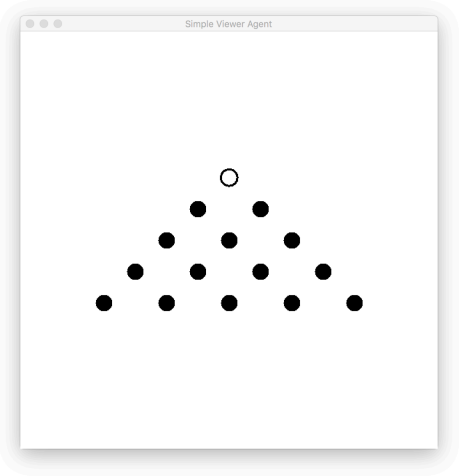
```
bash-3.2$ clisp
...

[1]> ( load "sva_pegs_demo.l" )
;; Loading file sva_pegs_demo.l ...
;;  Loading file simple_viewer_agent.l ...
;;  Loaded file simple_viewer_agent.l
;; Loaded file sva_pegs_demo.l
T
[2]> ( peg-demo )
NIL
[3]>
```

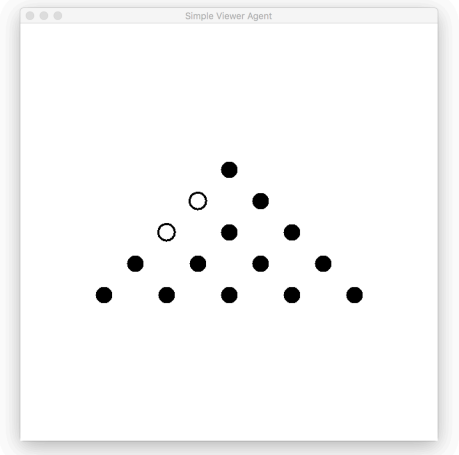
Textual Output from the Java Program

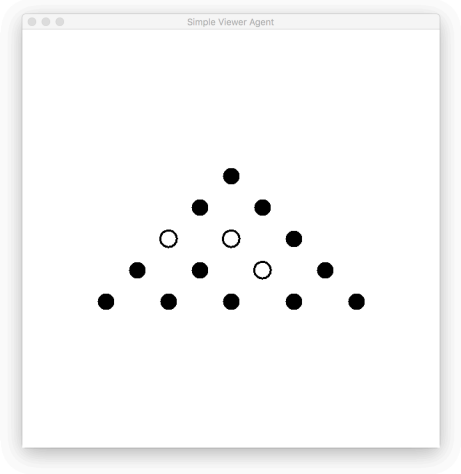
```
bash-3.2$ java -jar SimpleViewerAgent.jar
SVA: message = INIT 11 HIDE 20
SVA: message = MODE CIRCLE
SVA: message = DRAW 8 6 B
SVA: message = PAINT 7 5 B 7 7 B
SVA: message = PAINT 6 4 B 6 6 B 6 8 B
SVA: message = PAINT 5 3 B 5 5 B 5 7 B 5 9 B
SVA: message = PAINT 4 2 B 4 4 B 4 6 B 4 8 B 4 10 B
SVA: message = PAINT 6 4 W
SVA: message = DRAW 6 4 B
SVA: message = PAINT 7 5 W
SVA: message = DRAW 7 5 B
SVA: message = DRAW 8 6 W
SVA: message = PAINT 8 6 B
SVA: message = PAINT 5 7 W
SVA: message = DRAW 5 7 B
SVA: message = PAINT 6 6 W
SVA: message = DRAW 6 6 B
SVA: message = DRAW 7 5 W
SVA: message = PAINT 7 5 B
```

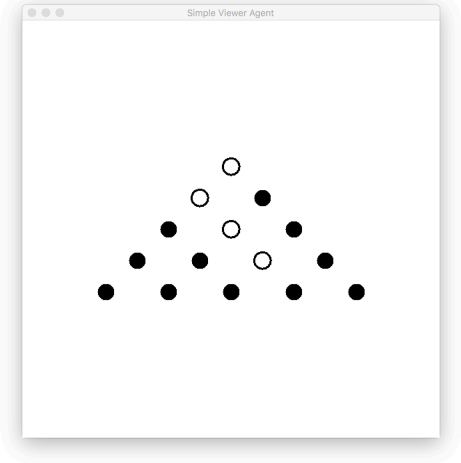
```
SVA: message = PAINT 8 6 W
SVA: message = DRAW 8 6 B
SVA: message = PAINT 7 5 W
SVA: message = DRAW 7 5 B
SVA: message = DRAW 6 4 W
SVA: message = PAINT 6 4 B
SVA: message = PAINT 6 8 W
SVA: message = DRAW 6 8 B
SVA: message = PAINT 7 7 W
SVA: message = DRAW 7 7 B
SVA: message = DRAW 8 6 W
SVA: message = PAINT 8 6 B
SVA: message = PAINT 5 3 W
SVA: message = DRAW 5 3 B
SVA: message = PAINT 6 4 W
SVA: message = DRAW 6 4 B
SVA: message = DRAW 7 5 W
SVA: message = PAINT 7 5 B
```

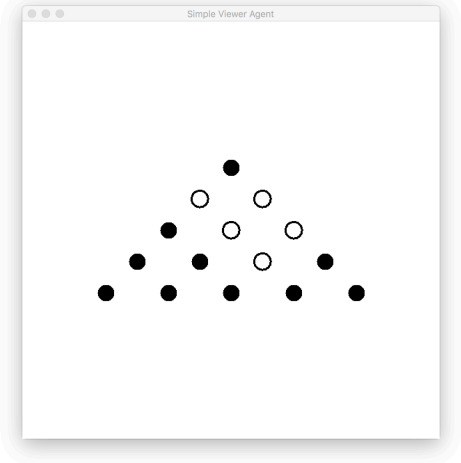
Snapshots of the Graphical Output

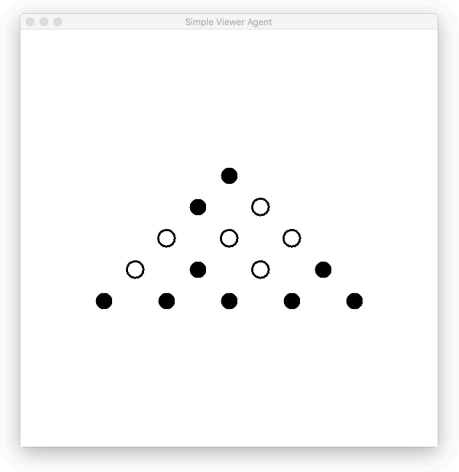





Simple Viewer Agent


Simple Viewer Agent


Simple Viewer Agent



sva_pegs_demo.l

```
;;;; File: sva_dobo_demo.l
;;;; Line: Peg solitaire demo of the Lisp API for doing just a few very simple visual things

;; IMPORTS

( load "simple_viewer_agent.l" )

;; THE DOBO DEMO OF sva.l

( defun peg-demo ()
  ( send ( init ( list 11 'hide 20 ) ) )
  ( send ( mode 'circle ) )
  ( setup-board ) ( sleep 5 )
  ( first-move ) ( sleep 5 )
  ( second-move ) ( sleep 5 )
  ( third-move ) ( sleep 5 )
  ( fourth-move ) ( sleep 5 )
  ( fifth-move ) ( sleep 5 )
  nil
)

( defun setup-board ()
  ( send ( draw '( ( 8 6 B ) ) ) )
  ( send ( paint '( ( 7 5 B ) ( 7 7 B ) ) ) )
  ( send ( paint '( ( 6 4 B ) ( 6 6 B ) ( 6 8 B ) ) ) )
  ( send ( paint '( ( 5 3 B ) ( 5 5 B ) ( 5 7 B ) ( 5 9 B ) ) ) )
  ( send ( paint '( ( 4 2 B ) ( 4 4 B ) ( 4 6 B ) ( 4 8 B ) ( 4 10 B ) ) ) )
  nil
)

( defun first-move ()
  ( send ( paint '( ( 6 4 W ) ) ) ) ( send ( draw '( ( 6 4 B ) ) ) )
```

```
  ( send ( paint '( ( 7 5 W ) ) ) ) ( send ( draw '( ( 7 5 B ) ) ) )
  ( send ( draw '( ( 8 6 W ) ) ) ) ( send ( paint '( ( 8 6 B ) ) ) )
)

( defun second-move ()
  ( send ( paint '( ( 5 7 W ) ) ) ) ( send ( draw '( ( 5 7 B ) ) ) )
  ( send ( paint '( ( 6 6 W ) ) ) ) ( send ( draw '( ( 6 6 B ) ) ) )
  ( send ( draw '( ( 7 5 W ) ) ) ) ( send ( paint '( ( 7 5 B ) ) ) )
)

( defun third-move ()
  ( send ( paint '( ( 8 6 W ) ) ) ) ( send ( draw '( ( 8 6 B ) ) ) )
  ( send ( paint '( ( 7 5 W ) ) ) ) ( send ( draw '( ( 7 5 B ) ) ) )
  ( send ( draw '( ( 6 4 W ) ) ) ) ( send ( paint '( ( 6 4 B ) ) ) )
)

( defun fourth-move ()
  ( send ( paint '( ( 6 8 W ) ) ) ) ( send ( draw '( ( 6 8 B ) ) ) )
  ( send ( paint '( ( 7 7 W ) ) ) ) ( send ( draw '( ( 7 7 B ) ) ) )
  ( send ( draw '( ( 8 6 W ) ) ) ) ( send ( paint '( ( 8 6 B ) ) ) )
)

( defun fifth-move ()
  ( send ( paint '( ( 5 3 W ) ) ) ) ( send ( draw '( ( 5 3 B ) ) ) )
  ( send ( paint '( ( 6 4 W ) ) ) ) ( send ( draw '( ( 6 4 B ) ) ) )
  ( send ( draw '( ( 7 5 W ) ) ) ) ( send ( paint '( ( 7 5 B ) ) ) )
)
```

10 Example: Rendering a Carpet Board

The Scenario

The `sva_carpet_demo.l` file contains a demo for displaying a carpet board. It makes use of `sva.l`, the Lisp API for rendering graphics, which itself leans on the Java program `SimpleViewerAgent.java` which is contained in `SimpleViewerAgent.jar`.

Lisp Session

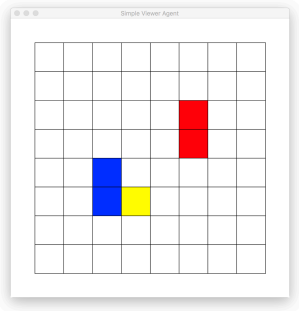
```
bash-3.2$ clisp
...

[1]> ( load "sva_carpet_demo.l" )
;; Loading file sva_carpet_demo.l ...
;;  Loading file simple_viewer_agent.l ...
;;  Loaded file simple_viewer_agent.l
;; Loaded file sva_carpet_demo.l
T
[2]> ( carpet-demo )
NIL
[3]>
```

Textual Output from the Java Program

```
bash-3.2$ java -jar SimpleViewerAgent.jar

SVA:  I found a file.
SVA: command = init
SVA: line = 8 SHOW
I will try to initialize ...

SVA:  I found a file.
SVA: command = mode
SVA: line = SQUARE
I will try to change mode ...

SVA:  I found a file.
SVA: command = draw
SVA: line = 5 6 R 6 6 R
I will try to paint what I found ...

SVA:  I found a file.
SVA: command = draw
SVA: line = 3 3 Y 3 4 Y
```

```
I will try to paint what I found ...

SVA:  I found a file.
SVA: command = draw
SVA: line = 3 3 I 4 3 I
I will try to paint what I found ...
```

Snapshots of the Graphical Output



sva_carpet_demo.l

```
;;;; File: sva_carpet_demo.l
;;;; Line: Carpet demo of the Lisp API for doing just a few very simple visual things

;; IMPORTS

( load 'simple_viewer_agent.l )

;; THE CARPETO DEMO OF sva.l

( defun carpet-demo ()
  ( send ( init '(8 show) ) )
  ( send ( mode 'square ) )
  ( send ( draw '( ( 5 6 R ) ( 6 6 R) ) ) )
  ( send ( draw '( ( 3 3 Y ) ( 3 4 y) ) ) )
  ( send ( draw '( ( 3 3 i ) ( 4 3 i) ) ) )
  nil
)
```